CSE 391

Git branches, merging
Remote repositories

Slides created by Josh Ervin. Based off slides made by Marty Stepp, Jessica Miller, Ruth Anderson, Brett Wortzman, and Zorah Fung

AGENDA

- Logistics, be sure to check your homework scores
- More `git`
  - Branching
  - Merging
  - Working with remote repositories

ROADMAP

- Introduction to the command line
- Input/output redirection, pipes
- More input/output redirection, tee, xargs
- Git: Fundamentals
- Git: Branches, merging, and remote repositories
- Regular expressions
- More regular expressions, sed
- Users and permissions
- Bash scripting
- Industry applications

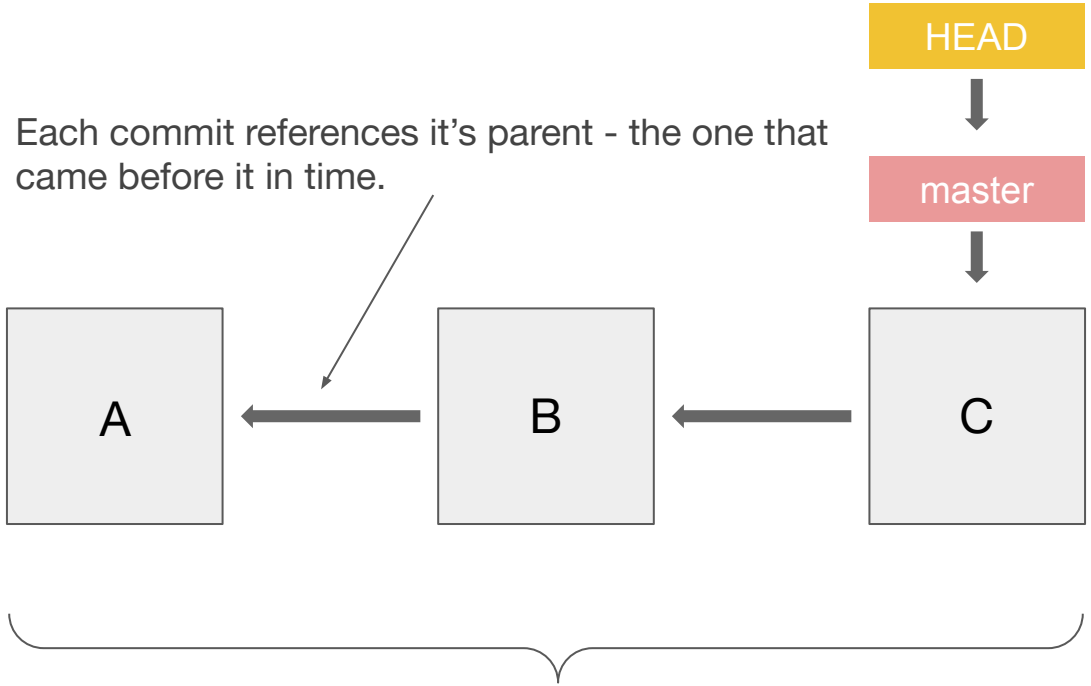
REVIEW: FOUR PHASES

Working Directory	Staging Area/Index	Local Repository	Remote Repository
Working changes	Changes you're preparing to commit	Local copy of the repository with your committed changes	Remote shared repository (Usually stored with a platform like GitHub/Gitlab)

GIT: BRANCHING

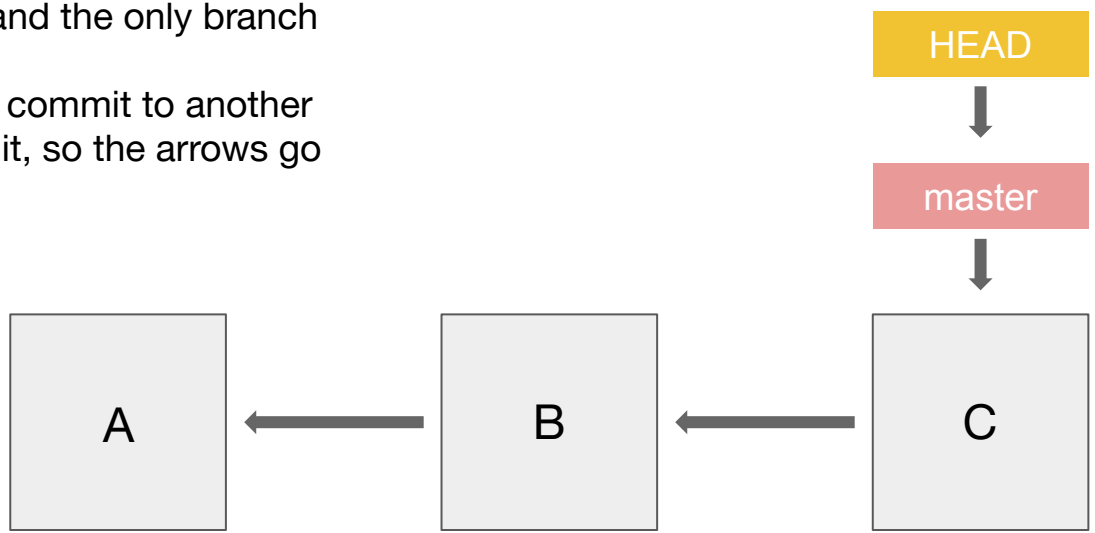
- So far, all the operations we've done have been on the master branch.
- However, it's very rare you'll be working on master directly. Instead, you'll work on a separate branch:
  - Master is the "single source of truth" - the history of the project
  - The code on master should be stable and compile
  - Because of this, it's difficult to share and collaborate on in progress work.
  - Working directly on master makes it difficult to work simultaneously on unrelated features.

COMMIT HISTORY



COMMIT HISTORY

- HEAD: This is a pointer, or reference, to the git branch that you are currently working on.
- Master: This the main branch of your repository, the single “source of truth”, and the only branch created by default.
- **NOTE:** The pointer from one commit to another points to the previous commit, so the arrows go back in time.



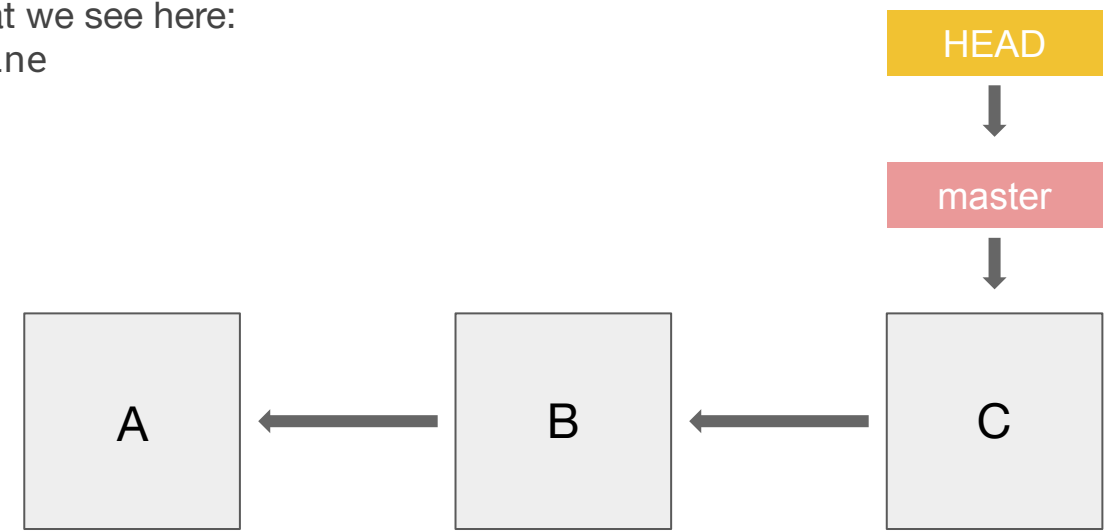
COMMIT HISTORY

To view this same information on the command line:

```
$ git log
```

Or, to make it more similar to what we see here:

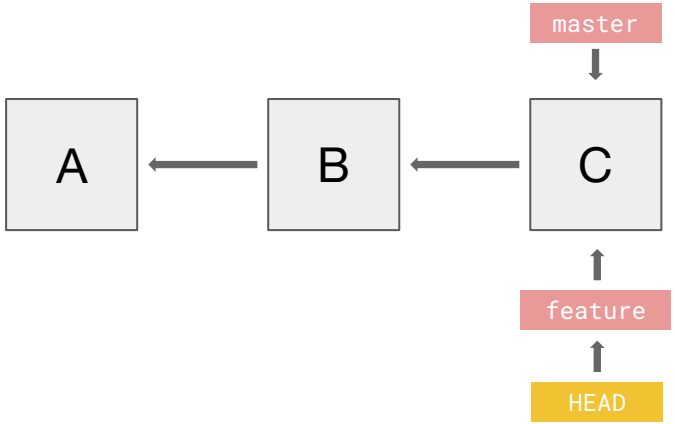
```
$ git log --graph --oneline
```



BRANCHING

Commands that have been run:

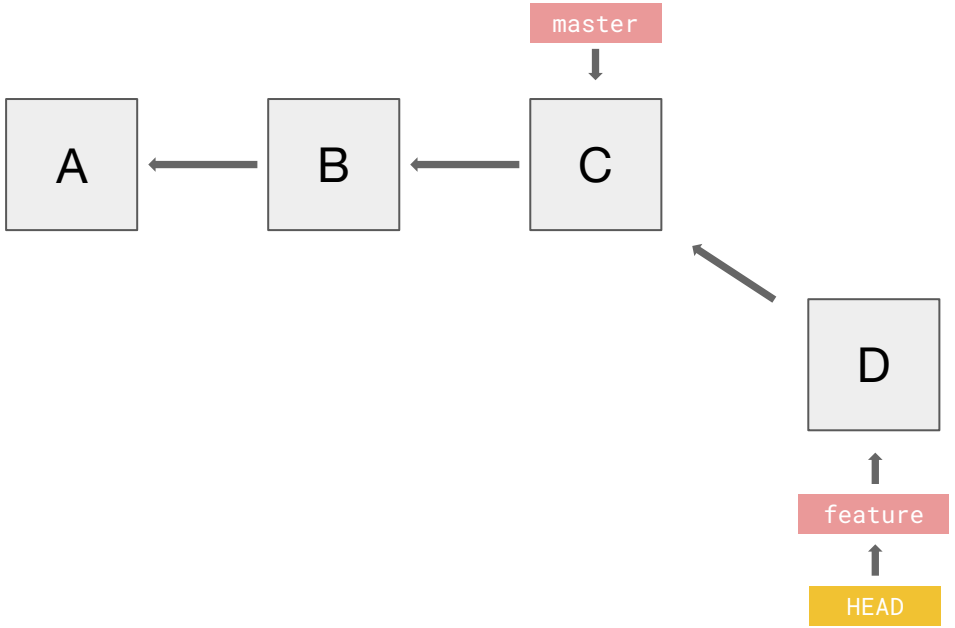
```
$ git branch feature
$ git checkout feature
```



BRANCHING

Commands that have been run:

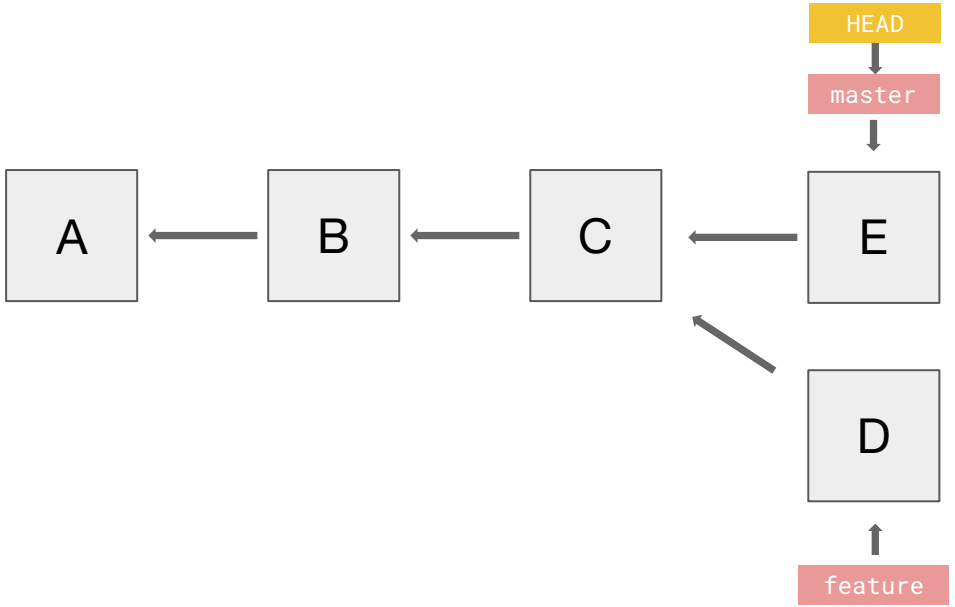
```
$ git branch feature
$ git checkout feature
$ echo “D” >> file.txt
$ git add file.txt
$ git commit -m “D”
```



BRANCHING

Commands that have been run:

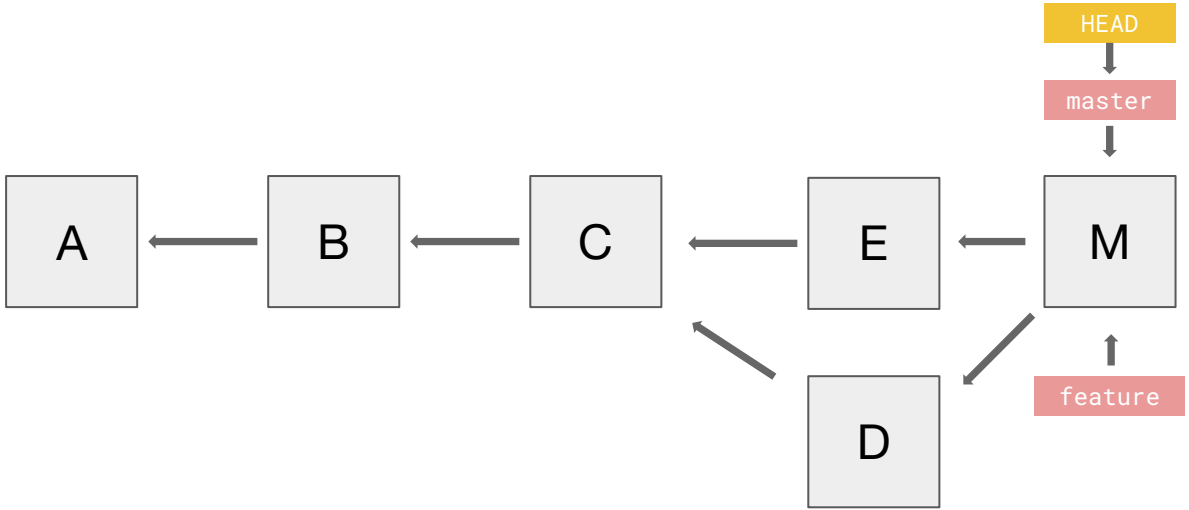
```
$ git branch feature
$ git checkout feature
$ echo “D” >> file.txt
$ git add file.txt
$ git commit -m “D”
$ git checkout master
$ echo “E” >> file.txt
$ git add file.txt
$ git commit -m “E”
```



BRANCHING

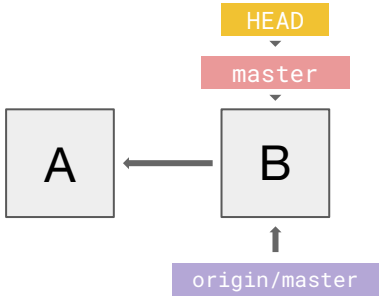
Commands that have been run:

```
$ git branch feature
$ git checkout feature
$ echo “D” >> file.txt
$ git add file.txt
$ git commit -m “D”
$ git checkout master
$ echo “E” >> file.txt
$ git add file.txt
$ git commit -m “E”
$ git merge
```

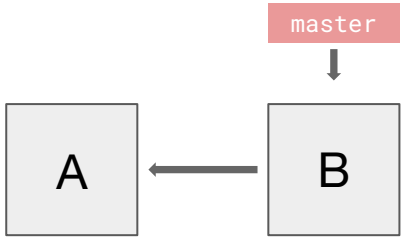


WORKING WITH REMOTE

Local

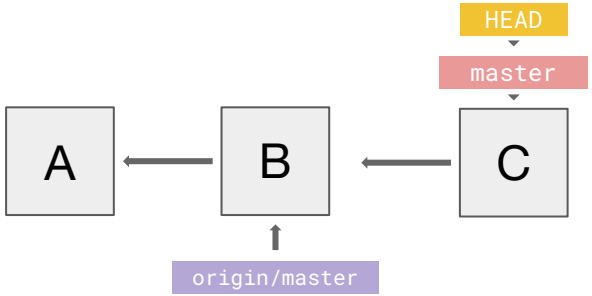


Remote

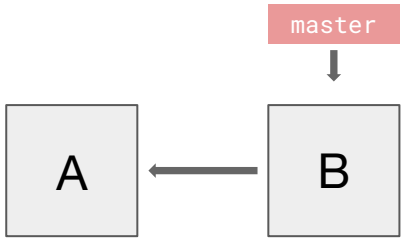


WORKING WITH REMOTE

Local

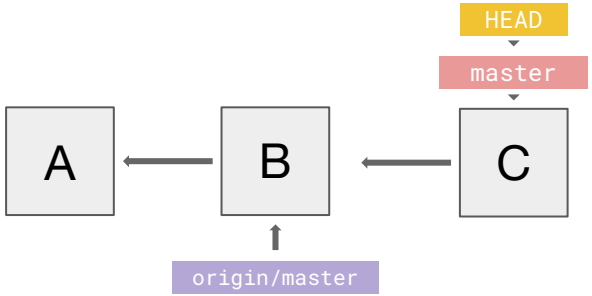


Remote

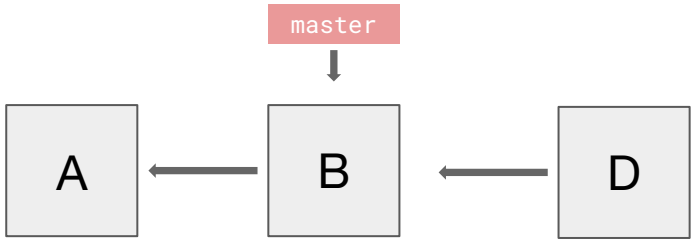


WORKING WITH REMOTE

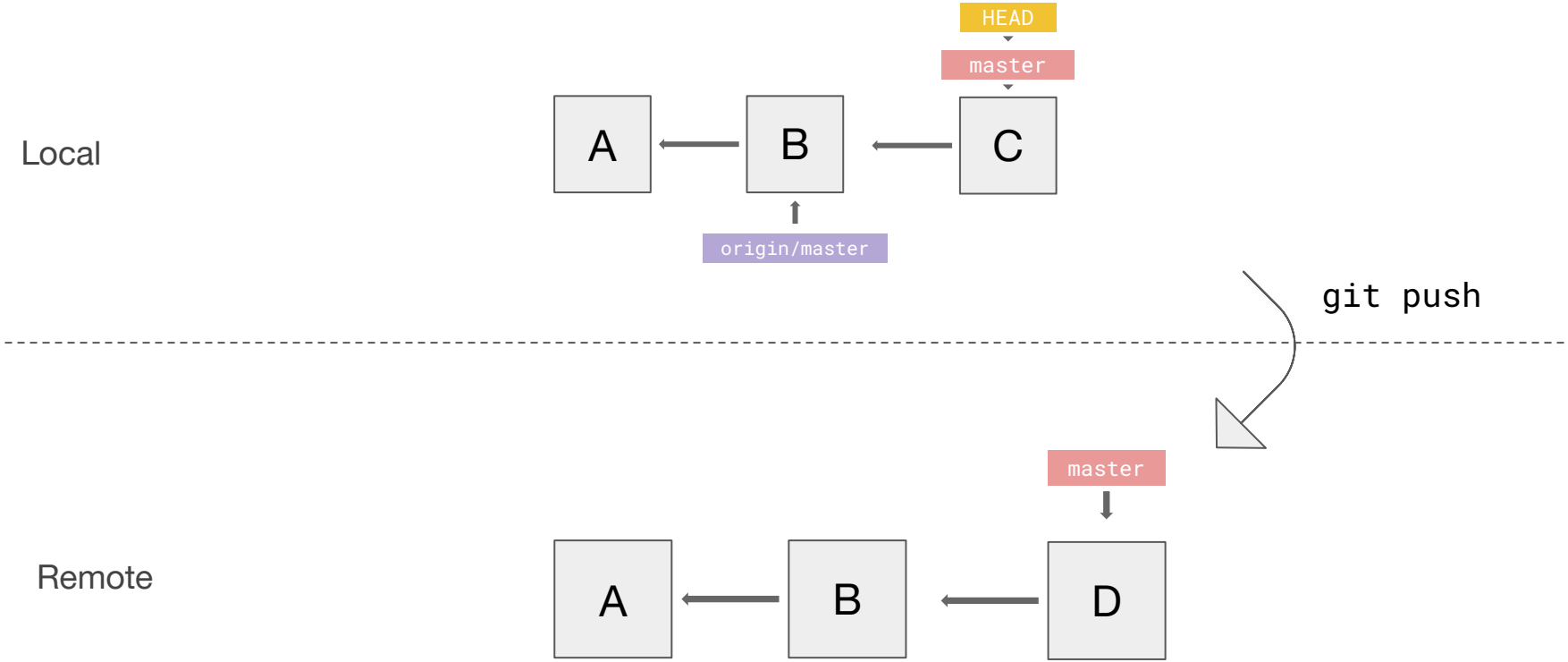
Local



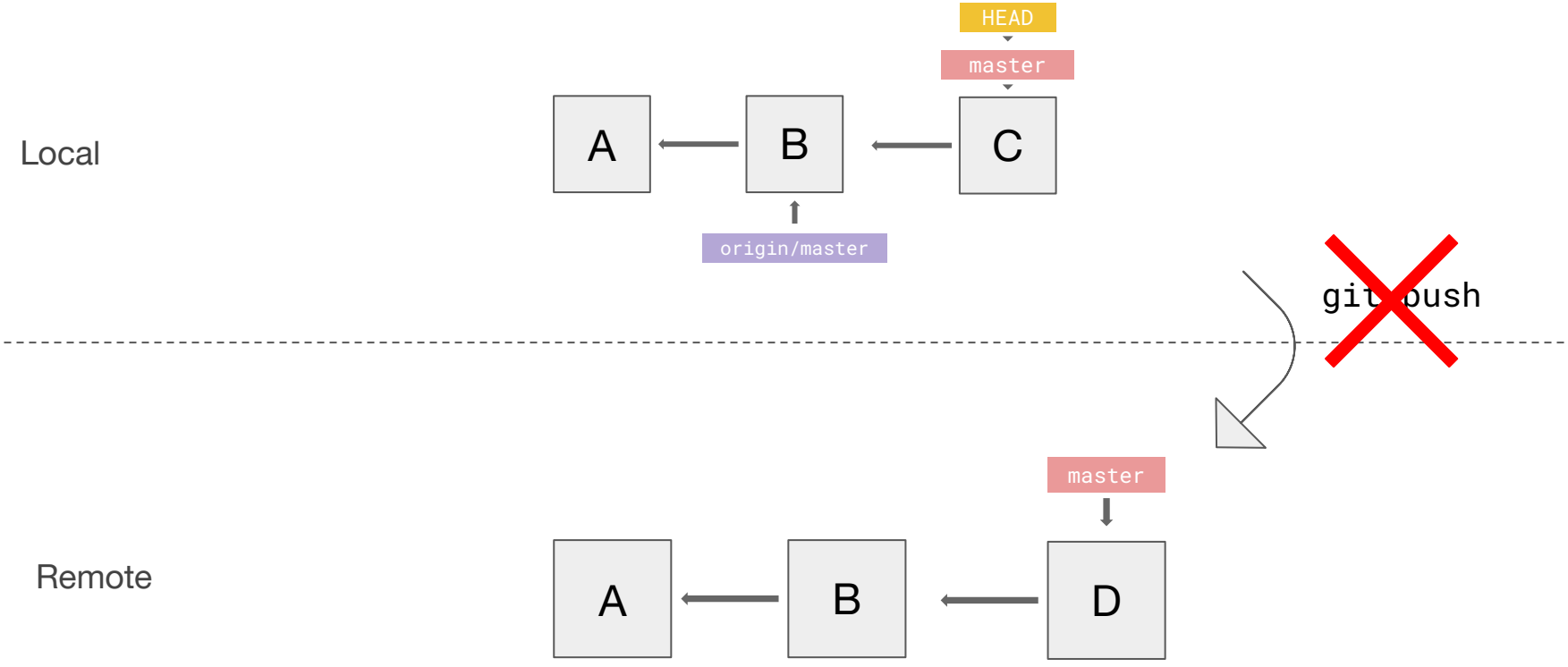
Remote



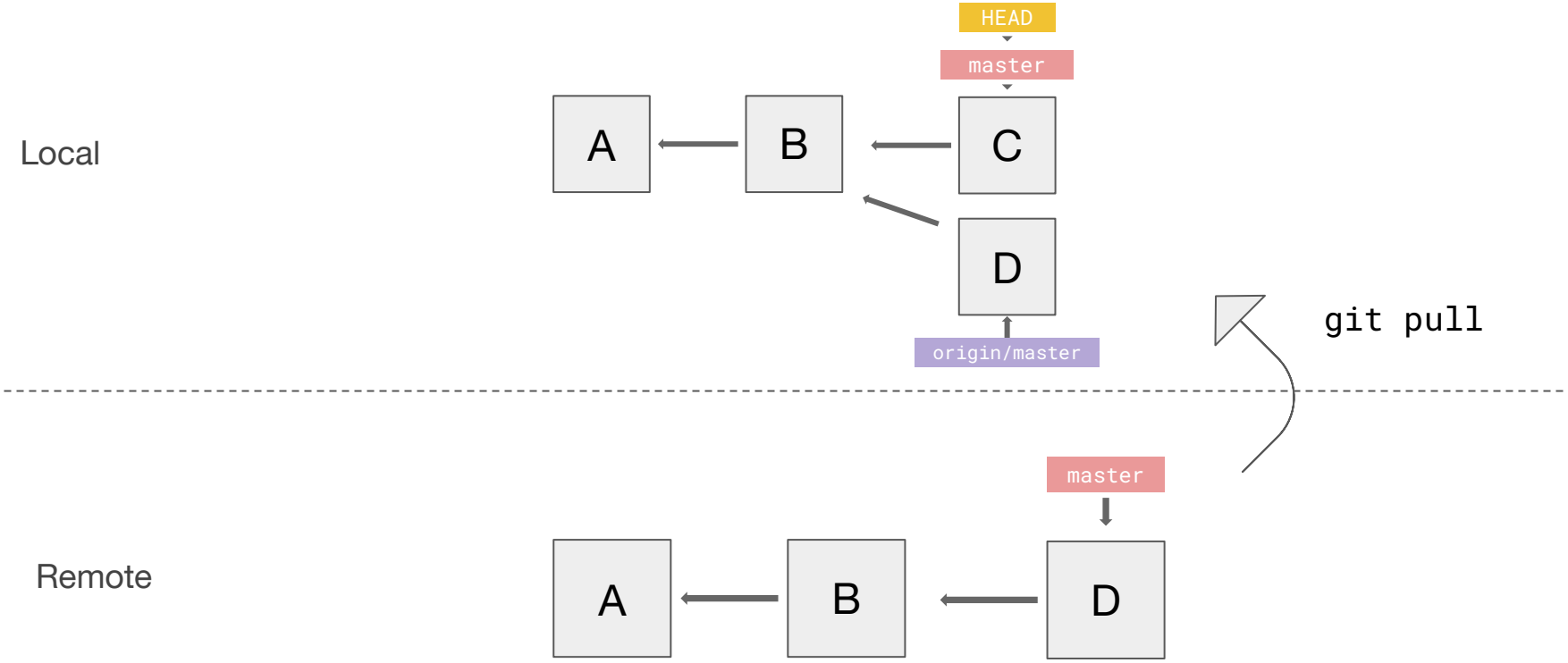
WORKING WITH REMOTE



WORKING WITH REMOTE

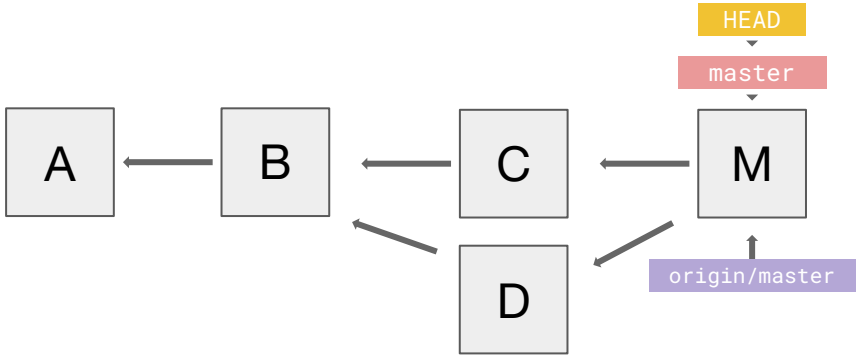


WORKING WITH REMOTE



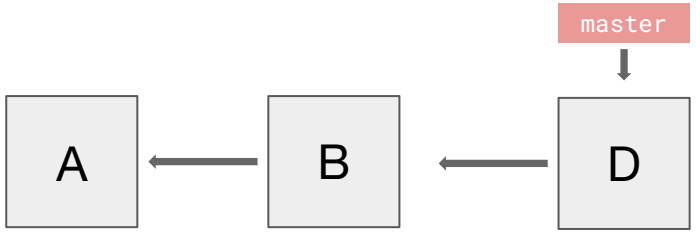
WORKING WITH REMOTE

Local

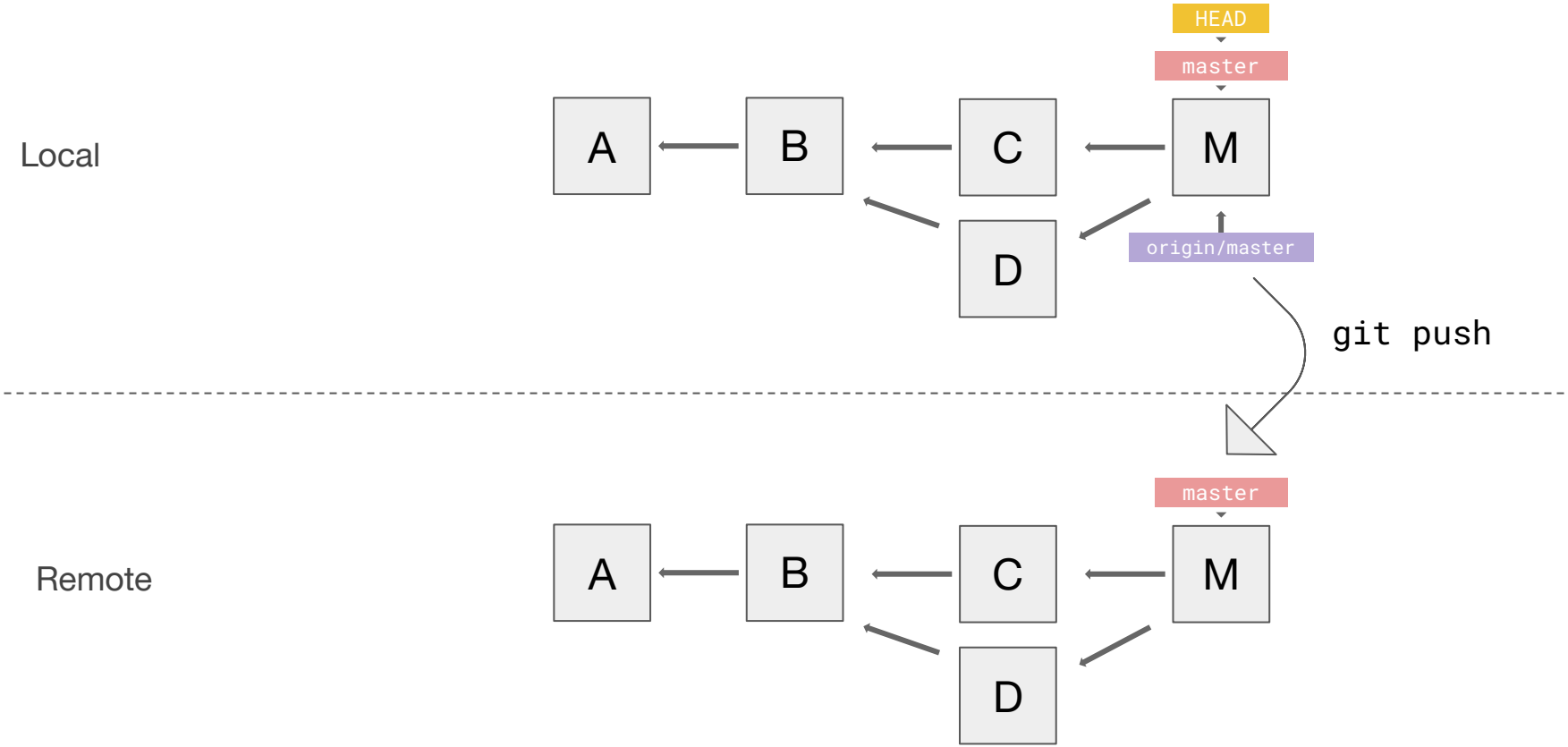


`git merge`

Remote



WORKING WITH REMOTE



THINK:

Suppose Josh and Hunter are both working on the ‘abc’ repository. The following sequence of events happen:

1) Josh creates a file named “lmno”, adds it to the index, commits it, and pushes it
2) Hunter creates a file named “icu”, adds it to the index, commits it, and pushes it

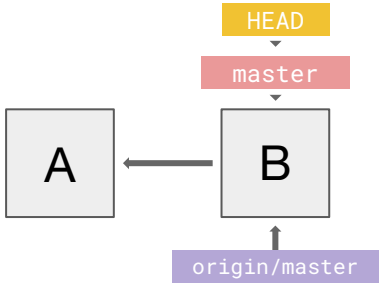
What happens on Hunter’s computer? Does it tell him that the push was successful? Or did the push fail?

PULL REQUESTS

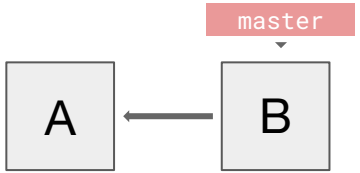
- In practice, it's very rare that we would merge branches locally and push them to remote.
- Instead, we use a service like GitHub or GitLab to help us:
  - Step 1: Create a local branch and make some commits
  - Step 2: Push those commits to remote
  - Step 3: Open a pull/merge request on GitLab
  - Step 4: Collaborate with others, leave comments, and fix conflicts
  - Step 5: Merge into master (or other branch)
- BUT, it's important to know what's going on when you branch and merge, because that's what's going on under the hood on GitHub/GitLab
- **The main goal here is to be very deliberate about what we put on master.**

WORKING WITH REMOTE

Local

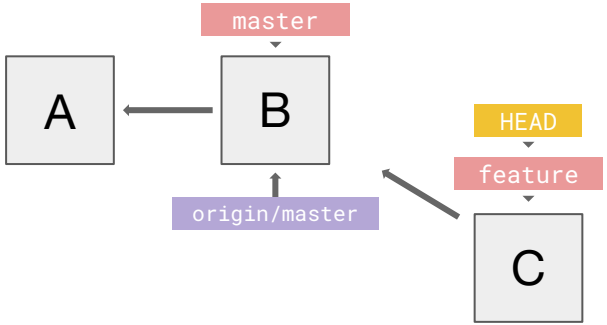


Remote

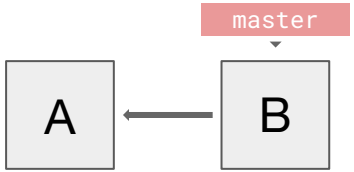


WORKING WITH REMOTE

Local

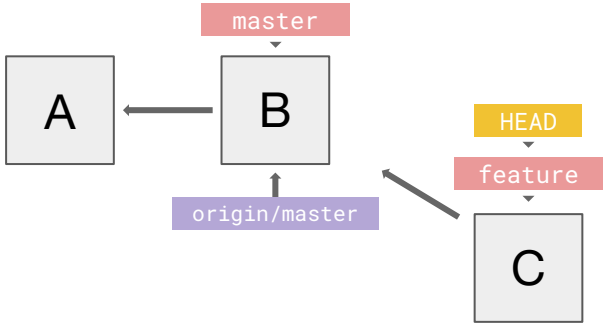


Remote

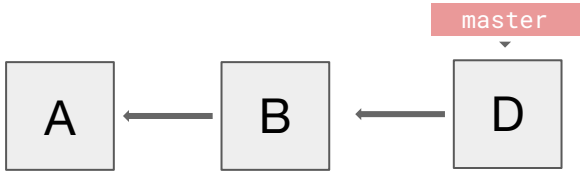


WORKING WITH REMOTE

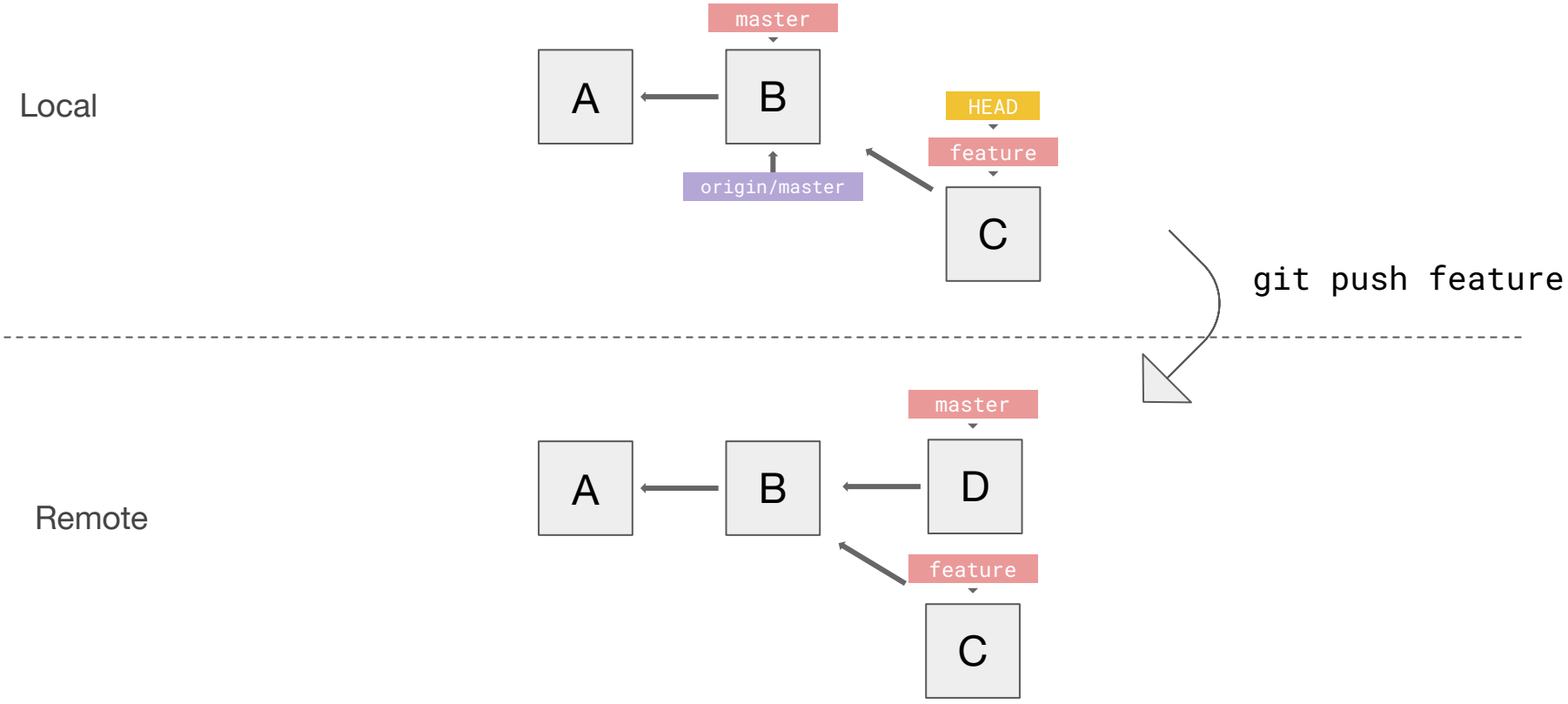
Local



Remote

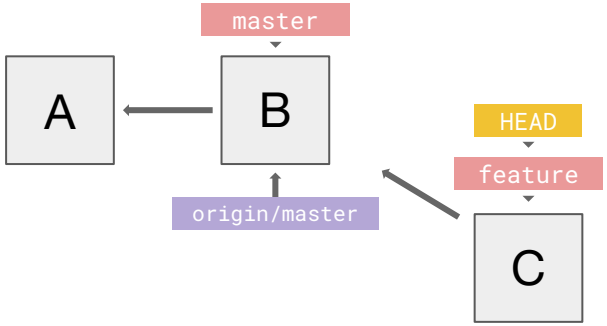


WORKING WITH REMOTE

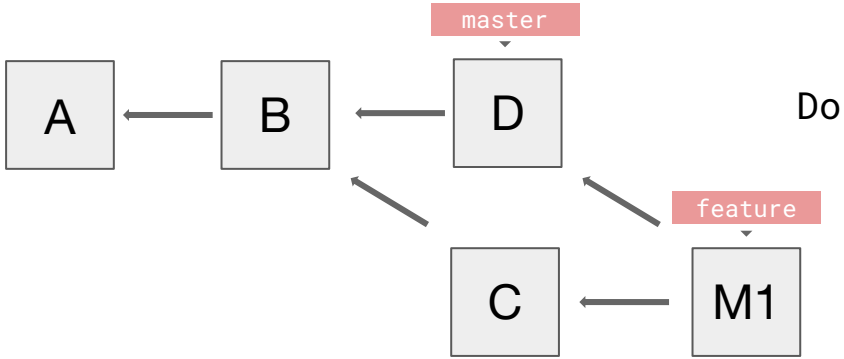


WORKING WITH REMOTE

Local



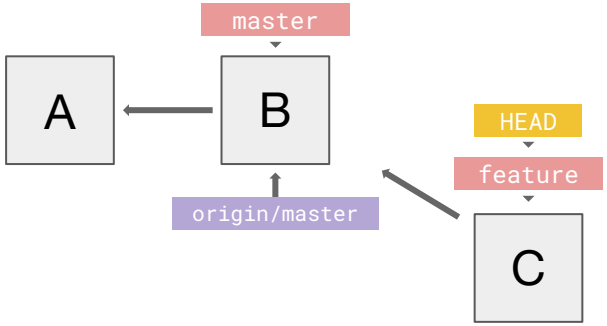
Remote



Do this on GitLab

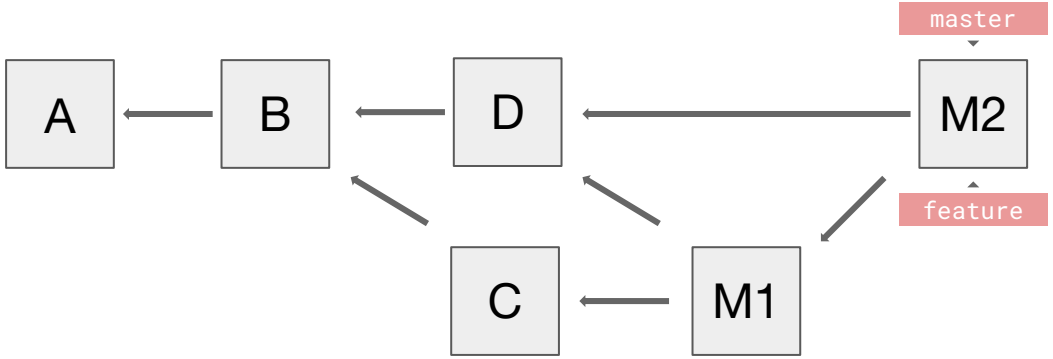
WORKING WITH REMOTE

Local



PULL REQUEST!!!

Remote

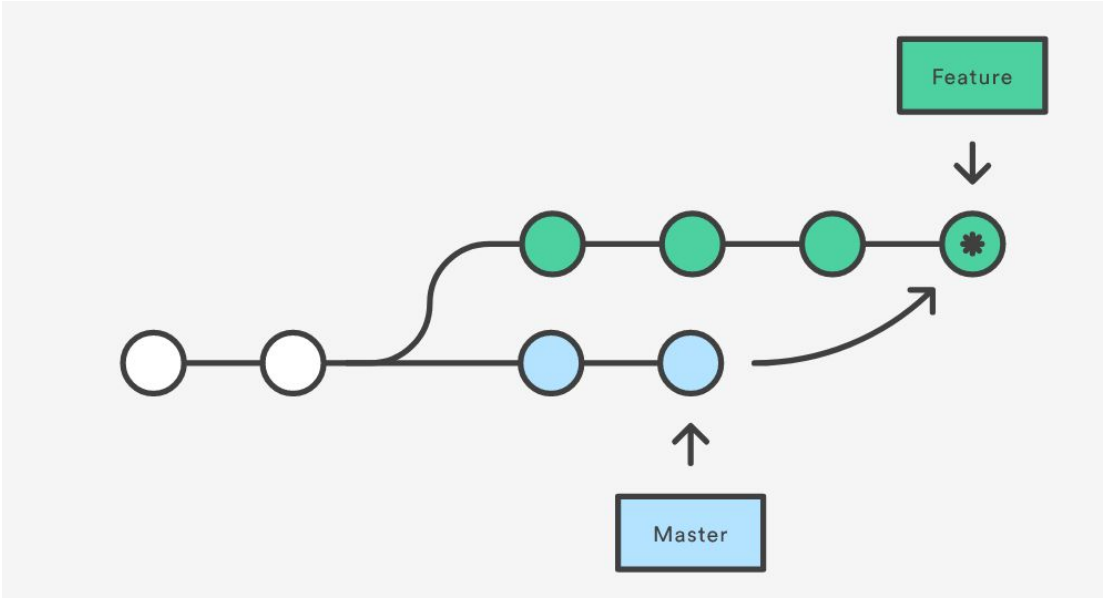


That’s all the Git you should need in daily life!

Each company might have their own custom system, but same principles apply.

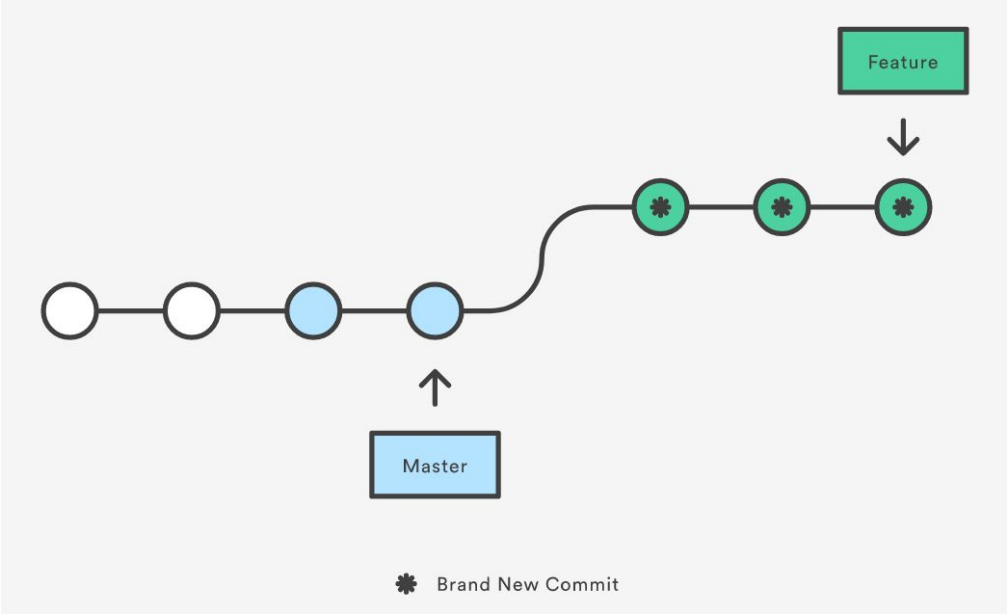
REBASE

- Generally we prefer to resolve diverging branches in the repository by merging the keepers
- This means we are always moving the branches forward in time:



REBASE

- There is another option, where instead we choose to rewrite the history of the repo
- This is the **`git rebase`** command



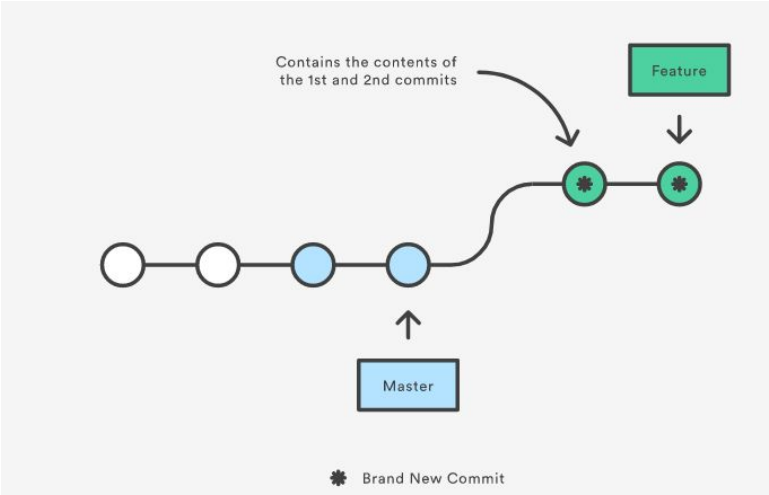
REBASE

- Two ways to use rebase:
  - Move entire feature branch to the end of the master branch (previous slide)

```
git checkout feature
git rebase master
```

  - Pick and choose which commits go into the end of the branch (squashing the history)

```
git checkout feature
git rebase -i master
(change the annotations)
```



WHEN TO REBASE

- Generally it is best to use merge
- Rebase is typically used when an organization wants to create a cleaner history
- Generally one person will be in charge of rebasing the “official” history
- Never rebase on a project unless you are in charge or have been told to do so!